Facultat d'Informàtica de Barcelona

Universitat de Catalunya

Visualization of multiple Molecular Simulation paths

Author:
Miguel Ángel Élez-Villarroel Garcia

Supervisor:
Pere-Pau Vázquez Alcocer

A thesis submitted for the degree of

Master in Innovation and Research in Informatics

Computer Graphics and Virtual Reality

June 27, 2019

Abstract

Molecular Dynamics (MD) is a computer simulation method that studies the physical movements of atoms and molecules. For a simulation to be stable it requires little time steps, and each time step needs a substantial amount of computation. They are costly.

For that very reason they are often calculated in super-computers, calculating and obtaining dozens or hundreds of trajectories. Once they are obtained they need to be inspected, but the current methods do not allow for an inspection of the whole set at the same time, taking even longer

In this project we have developed a visualization system able to provide the necessary tools to have an overview of multiple trajectories, easily find patterns and provide information on detail about individual trajectories if needed.

Acknowledgements

It is usual to thank those individuals who have provided particularly useful assistance, technical or otherwise, during your project.

Contents

List of Figures

List of Tables

Chapter 1

Introduction

In the field of molecular modeling, docking is a method which predicts a preferred orientation of one molecule to a second when bound to form a stable complex (chemical system composed of a discrete number of molecules).

One of the fields more used is in structure-based drug dessign.Docking usually happens when the system is stable: this is usually when the total energy of the system is low but another reason for docking is when the ligand finds a cavity in the protein where it can do docking.

To check these conditions for each simulated path, given their huge number, would be a very time-consuming task.

Our objective is to create a HTML page where they could input the data and using different methods of visualization (divided in four different views) would be able to quickly grasp the content of the general data so it can focus its attention in some specific paths, reducing the number of candidates to analyze.

This project will be divided into 5 more chapters:

In chapter 2 we will explain the architecture of the project. How the HTML page will work, how will have access to the data and the which will be the dataset we will be using.

Chapter 3 will be a little overview about which visualizations we will use in the different views.

In Chapter 4 we will explain which data, which programming languages and libraries we will use and how the functionalities of the views are implemented internally.

Chapter 5 will explain clearly which are the different interactions inside one view and with respect to the other ones.

Finally, in chapter 6 we have the conclusions and future work.

Further information can be found here: `https://goo.gl/k2huN9`.

1.1 LaTeX code examples and formatting tips

Hello, here's a citation [**?**]. References are stored in a Bibtex file. Google Scholar and IEEExplore allow you to download citations of papers in Bibtex format from their search engine. Some people use JabRef (`http://www.jabref.org`) to manage their database of references.

This is an inline equation $\Gamma(t) = K_i e^{\sin^2(\omega_t)}$. The first paragraph appears without indent but the following ones will have an indentation.

This is an actual named equation:

$$v(x) = \frac{1}{2}\sin(2\omega t + \phi)e^{-jst} \tag{1.1}$$

where ω is the angular speed. Notice that symbols liks ω should be written in italics whereas measurement units such as V for Volts appear as normal text. This paragraph didn't have an indentation because the first sentence was linked to the definition of equation (1.1). A code snippet for an example program is shown in Listing 1.1.

Listing 1.1: Source code for *hello.m*

```
for i:=maxint to 0 do
begin
{ do nothing }
end;
Write('Case insensitive ');
Write('Pascal keywords.');
```

The characteristic parameters of the system are sumarised in Table 1.1. A figure is shown Fig 1.1, we don't necessarily know if this figure will appear below, above or elsewhere; therefore, the text should never refer to the figure with sentences such as *"As shown here:"*.

Figure 1.1: Felix the Cat

Parameter	Value	Units
P	1	kW
Q	0	kVAr

Table 1.1: Characteristic parameters of the system

Sometimes, the symbols in an equation are defined as follows[1]:

$$V(t) = A\sin(\omega t + \theta_0) \tag{1.2}$$

where V is a voltage waveform,
A is the amplitude of the voltage,
ω is the angular frequency,
t is the time.

1.1.1 A brief comparison between a proper plot and a horrible plot

Figure 1.2 contains two plots of the same waveform. Subfigure 1.2(a) shows a badly formatted figure, Subfigure 1.2(b) shows a much better formatted figure. The problems with Subfigure 1.2(a), listed by order or relevance, are the following:

1. The font size is too small to be read properly.

[1]Some authors like to define their symbols this way.

2. The axes aren't labeled properly: the horizontal axis is not labeled and the units of the vertical axis are unknown. Further, symbols must be written in italics whereas numbers and units must be written as normal text.

3. The choice of limits for the axes is not good, the figure has wide useless empty spaces. The most relevant part of the waveform is the transient that happens between times t =0 and t =0.05 s, which is less than 10% of the timespan shown in the figure.

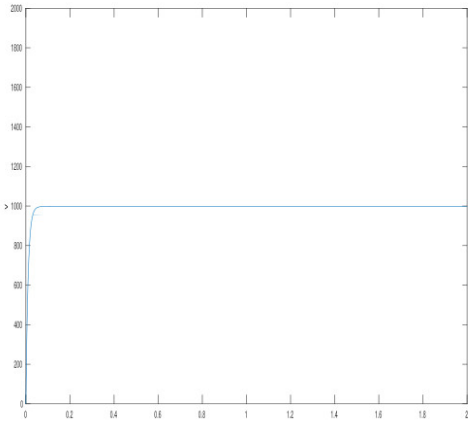
4. The figure has been scaled without keeping the original aspect ratio and fonts look narrower than they would if the figure had been scaled properly.

5. The plot doesn't have grid lines. This makes it hard to read the exact value (ie time, voltage) of points in the trace.

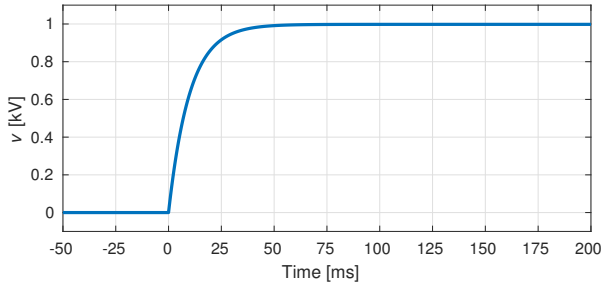
6. The width of the trace is too thin and may not be visible if printed in low resolution.

7. The choice of units of the vertical axis aren't the best. For example, in this case the plot would be easier to read if voltage had been expressed in kV instead of V.

8. The figure was exported as a bitmap (e.g. png, jpg, bmp) instead of being exported in vector format (e.g. eps, svg, pdf) and visual artifacts appear when the figure is scaled up or down in order to fit in the document.



(a) A horrible one.



(b) A proper one.

Figure 1.2: A figure with two subfigures.

Figure 1.3: Here's a large drawing of Felix the Cat that wouldn't fit in a portrait page

1.2 Objectives

1.3 Challenges

1.4 Contributions

Chapter 2

Architecture

We want our viewer to be able to work in a computer no matter which Operating System is on it. To achieve this, instead of implementing an executable program we will use browsers as presentation layer (front-end). This project will be developed in HTML+JavaScript, and tested in Google's Chrome web browser (last checked version 75.0.3770.100 (official build). As data access layer (back-end), instead of having a remote server with the data on it, we will have a localhost (the computer where the program will be executed will contain the needed data). To do this we will use Node.js.

2.1 Node.js

Node.js is an open-source, cross-platform JavaScript run-time environment that executes JavaScript code outside of a browser. Node.js lets developers use JavaScript to write command line tools and for server-side scripting. [X-PARA MIRAR].

We have to implement both, client-side and server-side. In Node.js the libraries are called modules and they are loaded using the require() function. The problem is that this function only works on server-side. This is solved using Browserify, a JavaScript tool that transforms the server-side modules into Node.js modules, being able to use from the browser all modules that were before only available for server-side.

To start our localhost we install http-server, a simple, zero-configuration command-line HTTP server. It is as simple as just writing "http-server" to make it work.

2.2 CORS

Cross-Origin Resource Sharing (CORS) is a mechanism that uses additional HTTP headers to tell a browser to let a web application running at one origin (domain) have permission to access selected resources from a server at a different origin.

For security reasons, browsers restrict cross-origin HTTP requests initiated from within scripts.

In our case, if we start the server depthmapCreator.html, some seconds after executing we will see this message in the Chrome Developer Tools.

Figure 2.1: Error when trying to execute deathmapCreator.html without CORS

To avoid it we install the package CORS and enable it by writing –cors in the http-server command.

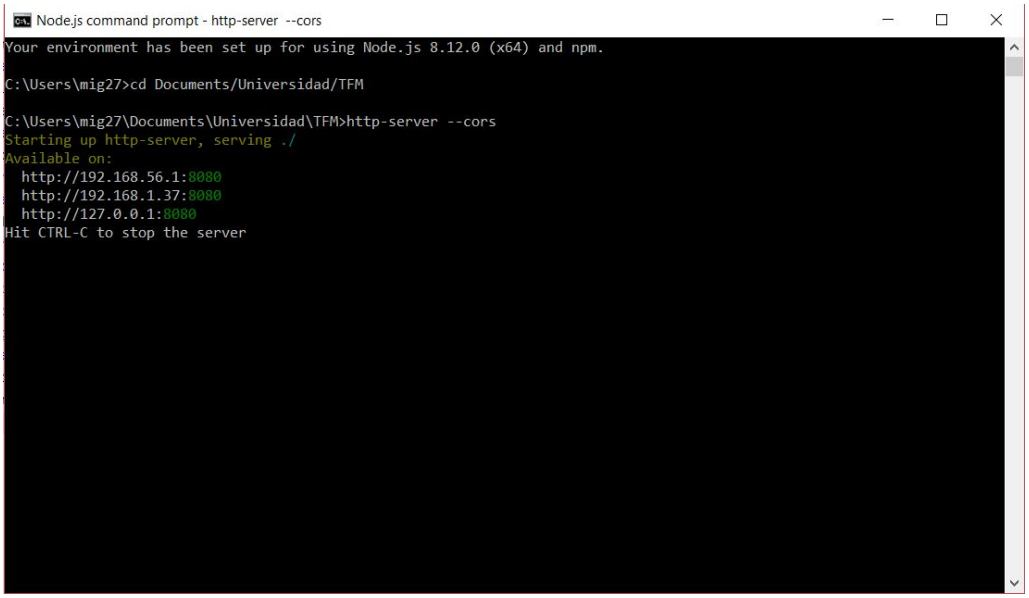


Figure 2.2: Executing http-server on Node.js command prompt

2.3 Data set

For this project we have two different types of data:

2.3.1 PDB

The Protein Data Bank (PDB) format provides a standard representation for macromolecular structure data derived from X-ray diffraction and NMR studies. This representation was created in the 1970's and a large amount of software using it has been written.

From the possible standard formats NGL is able to read a file and create a visualization, we were provided PDB format files.

PDB stands for Protein Data Bank and is a text file containing multiple lines. Each line of information is known as record and are composed by 80 columns, arranged in a specific order.

Protein Data Bank Format: Coordinate Section				
Record Type	**Columns**	**Data**	**Justification**	**Data Type**
ATOM	1-4	"ATOM"		character
	7-11	Atom serial number	right	integer
	13-16	Atom name	left–	character
	17	Alternate location indicator		character
	18-20	Residue name	right	character
	22	Chain identifier		character
	23-26	Residue sequence number	right	integer
	27	Code for insertions of residues		character
	31-38	X orthogonal Å coordinate	right	real (8.3)
	39-46	Y orthogonal Å coordinate	right	real (8.3)
	47-54	Z orthogonal Å coordinate	right	real (8.3)
	55-60	Occupancy	right	real (6.2)
	61-66	Temperature factor	right	real (6.2)
	73-76	Segment identifier	left	character
	77-78	Element symbol	right	character
	79-80	Charge		character
HETATM	1-6	"HETATM"		character
	7-80	same as ATOM records		
TER	1-3	"TER"		character
	7-11	Serial number	right	integer
	18-20	Residue name	right	character
	22	Chain identifier		character
	23-26	Residue sequence number	right	integer
	27	Code for insertions of residues		character

Table 2.1: Protein Data Bank Format

Following this structure, we find two differences with respect our provided files:

- Segment identifier (columns 73-76) is not present because it is obsolete. Even so, it is still used by some programs.

- TER only has the first field. TER record is used to indicate the end of a list of ATOM /HETATM records. The data that appears in the other columns indicates the last residue presented for each polypeptide and/or nucleic acid chain for which there are determined coordinates and has the same residue name, chain identifier, sequence number and insertion code as the terminal residue.

To animate this model, instead of having a trajectory file, we have multi-frame PDB files. Each PDB file contain multiple models. To indicate when and which model starts and when finishes we have two more record types:

Record Type	Columns	Data	Justification	Data Type
MODEL	1-5	"MODEL"		character
	11-14	Model serial number.	right	integer
ENDMDL		"ENDMDL"		

Table 2.2: Protein Data Bank Format

Each MODEL has its corresponding ENDMDL and all the models in a multi-model entry must represent the same structure.

2.3.2 run_trajectory_x

Text file consisting of:

- *Task*: Not relevant as it is always 1. Obsolete. Even so, it is still used by some programs.
- *Step*: Number of PELE step inside the simulation
- *numberOfAcceptedPeleSteps*: Number of accepted steps.
- *currentEnergy*: Total energy of the protein-ligand energy structure.
- *Binding energy*: protein-ligant interaction energy.
- *rmsd_lig*: Root-Mean-Square Deviation (distance) to the crystal that reflexes the binded structure.
- *sasaLig*: Solvent-accessible surface area. It measures how much of the ligand is exposed to the solvent.

2.3.3 Our project dataset

For this project we were given 50 folders (from 0 to 49) corresponding to 50 epochs (each epoch is a time step of variable duration) and 126 files. Those 126 files are divided into 2 groups: One group is run_trajectory_x.pdb and the other is run_trajectory_x, where x is a number between 1 and 63, representing processors.

Chapter 3

Overview

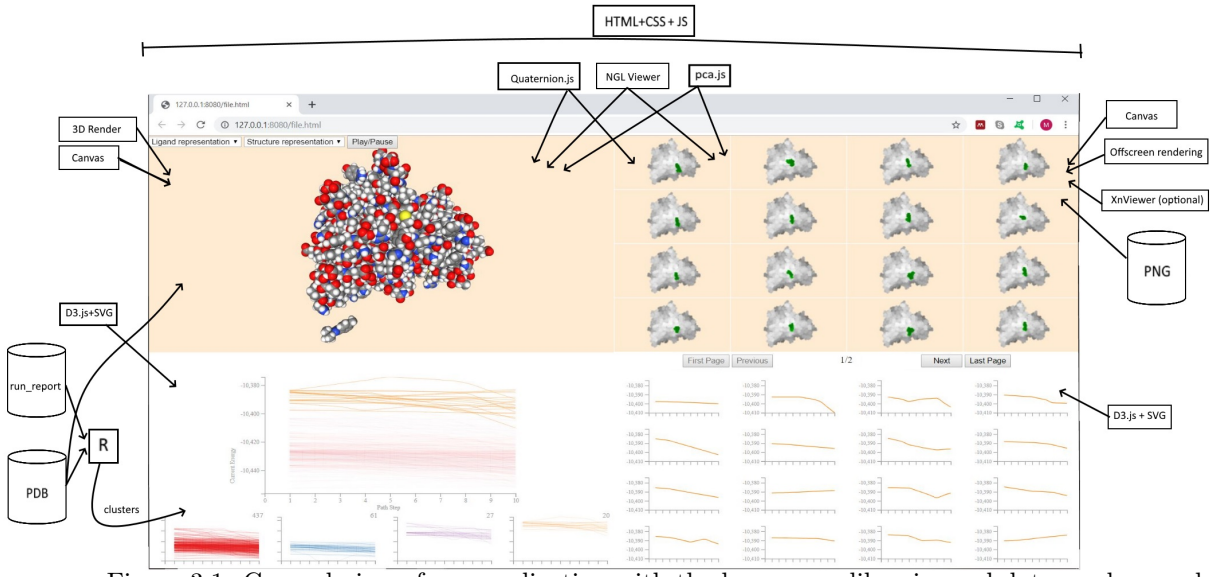


Figure 3.1: General view of our application with the languages, libraries and data used on each of the views

Using HTML and CSS the browser's screen is divided into four partitions. Starting from bottom-left to top-left, the four views help to discover patterns among the paths, depending on the total energy of the system and the position of the ligand with respect to the protein. With this the user could be able for example to discover in which situations a certain path do docking.

The four views are created to give information from a more general (clusters of multiple paths) to a more specific approach (individual paths) using different sort of visualizations. These are:

3D viewer (top-left): Contains a 3D rendered representation of the protein and ligand, letting us see with great detail the trajectory of an individual path, the position and name of the atoms and different types of representation. The model is loaded from a .pdb file using NGL Viewer over a canvas and the initial orientation is computed and applied using pca.js and quaternion.js.

Cluster visualization view (bottom-left): Using a line chart we are able to represent a huge number of paths at the same time and easily discover patterns among them. As said before those patterns will be caused by differences in the total energy of the system and the similarity among them. Those that are similar will be grouped into clusters, as they are similar enough to be grouped together. We decided only the four more populated clusters will be visible because the more samples, the more we can be sure there is a reason for them to be together (with small number of samples it can be a coincidence). The cluster will be created using an algorithm method called DBSCAN.

Depth map view (top-right): A depth map lets the user to know using colours how far an object is. Using small multiples, we can visualize up to 16 depth maps from one cluster in a 4x4 grid that will let us compare easily the position where each one is.

Individual plot view (bottom-right): 4x4 grid that contains the same paths as the cluster selected but instead of having them all in one plot we have one plot for each individual path. As in the depth map view, we can use see more specific data about the total energy of the multiples paths in one cluster and see patterns and differences.

The different visualizations, libraries and algorithms will be explain with grater detail in the next chapter.

Chapter 4

Implementation

4.1 3D viewer

As we said before, the 3D viewer will be our last view. The main objective of this view is to be able to observe the 3D representation of the protein-ligand model and be able to see its trajectory.

4.1.1 Input

- run_trajectory_x.pdb: PDB file of processor x.
- quaternion.txt: Text file that contains the initial orientation of the protein-ligand structure.

4.1.2 Languages and libraries

NGL viewer

Library and web application for visualization of macromolecular structures. Using mainly JavaScript and WebGL (CSS and HTML too for the web application GUI), the viewer can display and interact with large molecular models directly into the browser through the HTML5 canvas element without needing plug-ins.

As features we have:

- Molecular structures it can read: mmCIF, PDB, PQR, GRO, SDF, MOL2, MMTF.
- Multiple molecular representation types:
  - Some representations using cylinders and spheres for bonds and atoms are Ball-and-Stick, licorice and spacefill.
  - Some representations using secondary structure depictions based on backbone atoms are backbone, cartoon or ribbon.
- Density volumes it can read: MRC/MAP/CCP4, DX/DXBIN, CUBE, BRIX/DSN6, XPLOR/CNS.
- Some user interactions: mouse picking, zooming, animation, image export.
- Coordinate trajectories (DCD & PSF, NCTRAJ & PRMTOP, TRR/XTC & TOP, remote access via MDSrv)
- Embeddable (single file, API)

For animations you can have a molecular structure file that indicates the initial position and a coordinate trajectory file that would indicate how much each atom moves on each step of the animation or you can have a multi-model molecular structure, which contains the position of all atoms on each step.

Quaternion.js

JavaScript library for 3D rotations.

4.1.3 Implementation

To visualize the PDB file we use NGL viewer.

```
stage.loadFile("http://localhost:8080/dadesSimulacions/" + viewerEpoch + "/run_trajectory_" + viewerProcessor + ".pdb", {
  defaultRepresentation: true,
  asTrajectory: true
```

Figure 4.1: Start of the loading function

Using the *Stage* class we determine on which element we will create the scene. The scene will be created on a canvas, an HTML element used to draw graphics on it. Afterwards, we load the file using the *loadFile* function. As our PDB files are multi-frame (multiples models in the same file), and we want to be able to play the animation, we enable *asTrajectory*.

```
if (!changedRepresentation) stage.viewerControls.rotate(quaternion);
```

Figure 4.2: View rotation

We obtain the orientation from quaternion.txt and rotate the view with its values. How we obtain those values will be explained when talking about the Depth map view.

We use the class *TrajectoryPlayer* to animate the model. With player we will be able to play/pause the animation.

```
player = new NGL.TrajectoryPlayer(traj, {
  step: 1,
  timeout: 200,
  interpolateStep: 100,
  start: 0,
  end: traj.numframes,
  interpolateType: "linear"
});
```

Figure 4.3: Player definition

By default we will always have a 3D model loaded which will be the PDB of the first processor of the first epoch.

4.2 Cluster visualization view

Molecule simulations are not only time consuming but the number of data is very big too. To look for an specific case looking through all the data would take too long. Cluster visualization view uses an already implemented algorithm in R to gather all simulations that are alike. Thanks to this we do not need to go through all data but only in the one we can be interested.

4.2.1 Input

- run_trajectory_x.pdb: PDB file of processor x.
- run_report_x: Text file with data about each model of processor x.

4.2.2 Languages and libraries

R

Free software environment for statistical computing and graphics. With more than 14440 available packages, R has access a multiple already implemented useful functions that go from analyzing data to visualize it. We will use the fpc (Flexible Procedures for Clustering) package to apply the DBSCAN algorithm.

C++

With this programming language we will generate the next files:

- pdb.csv: It contains for each pair of paths the cosine similarity (explained in implementation), and the difference between means of current energies, binding energies, RMSD and SASA.
- distance.csv: For each pair of paths we compute distances based on similarity of the models and difference of the mean of current energies.
- pathReport.csv: File containing the mean of the reports for each path.
- runReport.csv: File containing all reports except for those paths that only have one model.
- interpolReport.csv: File containing paths with n models. The number of models is specified in the interpolationPoints variable inside the config.dat file.

D3.js

D3.js or simply D3 (stands for Data-Driven Documents) is a JavaScript library that allows the programmer to bind input data (what the programmer want to visualize) to arbitrary Document Object Model (DOM) elements, being able to produce dynamic and interactive data visualizations in web browsers.

4.2.3 Implementation

DBSCAN

In Machine Learning, we can classify the types of learning into three categories:

- Supervised learning
- Semi-supervised learning
- Unsupervised learning

Unsupervised learning helps to find unknown patterns in data sets without any pre-existing label, only looking at features.

One of the main methods of unsupervised learning is clustering; Clustering or cluster analysis in statistical data analysis and data mining is task of putting into the same group those objects that are similar in some sense.

There are multiple clustering methods, being K-Means one of them and DBSCAN another one of the most popular ones. K-Means is an easy to implement method but the main problem was that you needed to specify a number of groups and we do not know how many groups there are. Also, as we will see next, DBSCAN can avoid outliers but K-Means cannot. This would have mean another preprocess to, in some way, remove those outliers, even with the risk of not being really

one. For those reasons, even if DBSCAN needs another set of values anyway, we decided to choose DBSCAN.

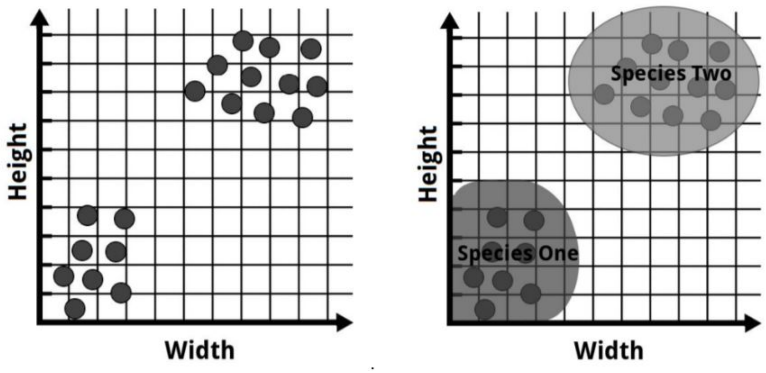


Figure 4.4: Left: unclassified data. Right: clustered data

Density-based spatial clustering of applications with noise (DBSCAN) is a data clustering algorithm proposed by Martin Ester, Hans-Peter Kriegel, Jörg Sander and Xiaowei Xu in 1996.

This algorithm identifies the clusters looking for high density areas separated by low density areas. We define this density as a minimum number of points within a specified radius. This method needs as input three parameters. Those are:

- dataset: contains the points data.
- Epsilon (eps): determines a specified radius. If there are enough points inside the neighbourhood of radius eps (called ϵ-neighborhood), it is considered a dense area.
- Minimum points (MinPts): Determines the minimum number of points we want in a neighbourhood to define a cluster.

DBSCAN works as seen on the next pseudocode:

```
DBSCAN(D, eps, MinPts)
   C = 0
   for each unvisited point P in dataset D
      mark P as visited
      NeighborPts = regionQuery(P, eps)
      if sizeof(NeighborPts) < MinPts
         mark P as NOISE
      else
         C = next cluster
         expandCluster(P, NeighborPts, C, eps, MinPts)
```

```
expandCluster(P, NeighborPts, C, eps, MinPts)
   add P to cluster C
   for each point P' in NeighborPts
      if P' is not visited
         mark P' as visited
         NeighborPts' = regionQuery(P', eps)
         if sizeof(NeighborPts') >= MinPts
            NeighborPts = NeighborPts joined with NeighborPts'
      if P' is not yet member of any cluster
         add P' to cluster C
```

```
regionQuery(P, eps)
   return all points within P's eps-neighborhood (including P)
```

Figure 4.5: DBSCAN pseudo-code

As seen before, the algorithm chooses one unvisited point from the dataset and checks the area around it. If the number of points (including the current one we are visiting) is bigger than MinPts then that point is called core point. In that case we can expand our cluster. Otherwise, we have two possibilities:

- The point we are visiting is already part of a cluster. That means it was a neighbour but there were not enough points for it to be core. In that case it is called border point.
- The point is not part of a cluster. Those points are noise.

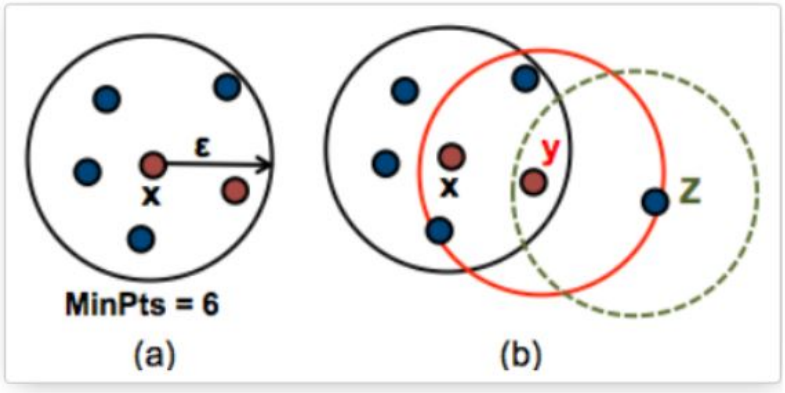


Figure 4.6: With distance being and $minPoints = 6$ x is a core point, y is a border point and z noise

With this we can observe the importance of choosing proper values for eps and MinPts. The former controls the local neighbourhood of the points. If it is too large, close clusters can merge and eventually be part of a single cluster. The latter primarily controls how tolerant the algorithm is towards noise (in large or noisy datasets can be desirable).

Cosine Similarity

The fpc package from R contains a DBSCAN implementation which needs the three aforementioned variables but instead of the data set we can pass directly a matrix with distances. This not only helps to fasten the execution of the algorithm but it let us decide which features we want to be more important too.

As we explained on the introduction, we are looking for situations where the model is likely to do docking. That happens when the system is stable. If a ligand is stable on a certain position it is highly possible that a model with a ligand and molecule on a similar situation is a stable system too. Having these two ideas in mind, we decided to create the next distance formula:

$$distance = alpha * similarity + (1 - alpha) * DiffCurrentEnergy \tag{4.1}$$

$$alpha = [0, 1], similarity = [0, 1], DiffCurrentEnergy = [0, 1]$$

As we do not want to give extra weight to one of the two sides (as DiffCurrentEnergy would be almost always higher than any similarity value and similarity could be negative), the variables are normalized. For each pair of paths, DiffCurrentEnergy is just the difference between both mean currentEnergy values. As the number of models is different depending on which path we choose how many models each path will have. Using the existing models inside each path we will interpolate as many intermediate values as we need. With this we can make comparisons 1:1.

For the similarity part, we will use the cosine similarity, which given two vectors of attributes, A and B, the cosine is represented using a dot product and a magnitude as

$$\text{similarity} = \cos(\theta) = \frac{\mathbf{A} \cdot \mathbf{B}}{\|\mathbf{A}\|\|\mathbf{B}\|} = \frac{\sum_{i=1}^{n} A_i B_i}{\sqrt{\sum_{i=1}^{n} A_i^2}\sqrt{\sum_{i=1}^{n} B_i^2}} \tag{4.2}$$

where Ai and Bi are components of vectors A and B respectively. In our case the vector will start at the centroid of the ligand and will finish at each of the atoms of the molecule. A and B will correspond to two vectors that finish at the same atom but on different paths. Once computed all similarities for each pair of paths we take the mean. The result will be a value between -1 and 1, being 1 when the two vectors point at the same direction, 0 when they point at perpendicular directions and -1 when pointing opposite directions. These values cannot be used directly into the distance function, as we want the similarity value to be 0 if they are the same and 1 if they are completely different. We can solve those problems by applying the next operation:

$$\text{finalsimilarity} = 1 - (\text{similarity} + 1)/2$$

(4.3)

With this the range passes from [-1,1] to [0,1] and the similarity will be 1 when they are completely different.
Lastly, the alpha value is going to be given by the user. Alpha determines which of the two parameters has more weight when applying the clustering algorithm, if it is the similarity between paths or the total energy of the path system. In figure XX we can observe an example of how we obtain the cluster file we use later to define the different groups in R. Among the groups showed, group 0 corresponds to noise points.

```
> library("fpc", lib.loc="~/R/win-library/3.5")
> distance <- read.csv("C:/Users/mig27/Documents/Universidad/TFM/distance.csv",header=FALSE)
> clustering <- fpc::dbscan(distance, eps = 0.4, MinPts = 10)
> print(clustering)
dbscan Pts=2726 MinPts=10 eps=0.4
         0  1  2  3    4
border 11  4  6  5    0
seed    0 23  6  8 2663
total  11 27 12 13 2663
> write.table(clustering$cluster,"C:/Users/mig27/Documents/Universidad/TFM/cluster.txt", row.names = FALSE)
```

Figure 4.7: R commands to execute DBSCAN and store the clusters

Finally, using D3.js we define a line chart where the y-axis is the value of total energy of the system (currentEnergy) and the x-axis is the number of steps. The number of steps is the same as the value of interpolationPoints in config.dat. The data it uses is from interpolReport.csv and the colour changes depending on the cluster. Only the four biggest clusters are shown in the plot and they are sorted by size from bigger to smaller.

One thing to notice is that even if the big plot has values and labels it is not the case for the little ones. Here we apply the data-ink ratio concept which was introduced by Edward Tufte, a pioneer in the field of data visualization.

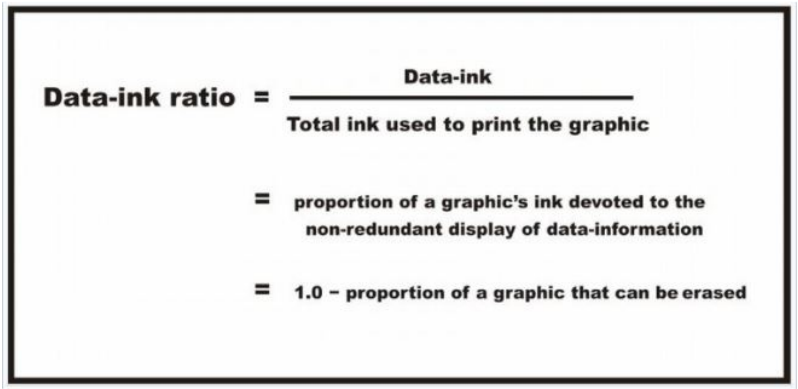


Figure 4.8: Data-ink ratio formula

The main idea is to take as many unnecessary elements as possible without removing elements needed for effective communication. In this case the big plot has the same values and labels as the small ones. Besides, the main intention for the small plots is to able to visualize patterns faster than checking them in the big plot, where the lines can be tangled.

4.3 Depth map view

Molecules have cavities where the ligand can enter and do docking. Using the depth map to see where the ligand is on each path and the small multiples (explained in 4.4.3), we can compare multiple paths at the same time using a 2D view. IN this section we will explained which files, libraries and how we implemented a web page to create PNG images that will contain the position of the ligand with respect to the protein even when they are not supposed to be visible (the ligand is inside or behind the protein).

4.3.1 Input

- pathReport.csv
- run_trajectory_x

4.3.2 Languages and libraries

- NGL Viewer
- Quaternion.js

UPNG.js

A small, fast and advanced PNG / APNG encoder and decoder.

A PNG image is composed at least by 4 different chunks. They are the IHDR Image header (must appear first), the PLTE colour palette, the IDAT image data and the IEND Image trailer (must appear last)

If we want to know which colour has each pixel of the image we need to decode the image beforehand.

PCA.js

JavaScript library used to compute Principal Components from a given matrix of data.

FileSaver.js

Library that lets you easily save into your hard disc blobs (Binary Large Objects), files or URLs (Uniform Resource Locators). It just needs to use the *saveAs()* function but you cannot specify a predefined download path. Instead the browser will ask you where do you want to download it or, if that option is disabled, they will automatically download to your default download location (usually the download folder but you can change its location in settings).

4.3.3 Implementation

PCA

Statistical procedure that uses an orthogonal transformation to convert a set of observations of possibly correlated variables (entities each of which takes on various numerical values) into a set of values of linearly uncorrelated variables called principal components. This transformation is defined in such a way that the first principal component has the largest possible variance (that is,

accounts for as much of the variability in the data as possible), and each succeeding component in turn has the highest variance possible under the constraint that it is orthogonal to the preceding components.

Knowing this we can use it to maximize the initial area we are going to show of the model. As we have 3 dimensions, what we really want is to minimize the distance between the nearest atom and the furthest in terms of depth, that is to say, variance. This variance is called eigenvalue and the direction is called eigenvector. Once we get which is the direction of the smallest eigenvalue we just need to take the other two eigenvectors as the other two axis, independent of which is which (the only important thing is that z-axis is the one with smaller variance). Once we get this new orientation we transform from a matrix into a quaternion. NGL Viewer has a function to change the orientation given a quaternion so we can apply it.

The protein is not expected to change shape that much, so we can take the first model (epoch 0, processor 1), get the orientation we are interested in and apply it to all the other models.

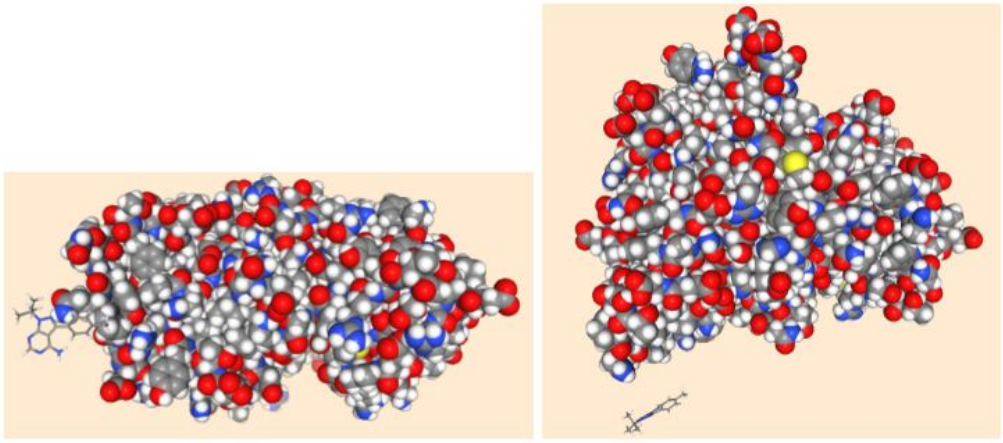


Figure 4.9: Comparison between original orientation (left) and after applying PCA (right)

The quaternion is stored in quaternion.txt, file that will be used for the default visualization on the 3D Viewer and the Depth map view.

Depth map

A depth map is an image or image channel that contains information about the distance between the surface of the objects from a given viewpoint.

First thing we need to compute our own depth map is to know which are the nearest and furthest point. These two correspond to the position of the nearest and furthest atom after applying the rotation. As each path has multiple models, we take the last one.

Once we have them, we decide to use a grey scale for the molecule and a red scale for the ligand; white/bright red color will be the nearest position and black/very dark red the furthest, leaving different tones of grey/red in between.

NGL Viewer has the *ColormakerRegistry* class that lets you, through the addScheme function, select a group of atoms and decide which colors they are going to have. We just need to create a function that defines the colour as we defined before, changing the base color from grey to red depending if the atom is a heteroatom or not. Even if this part is easy to implement there is a problem, shading (the depth map will have the proper colouring but with illumination applied). Shading helps us to perceive depth changing the colour depending for example on light source position or distance but in this case it can give us misleading information, as this modifies the values

we codified on each atom. To solve this we just need to modify the shader where the illumination is being computed, adding a boolean that determines if shading needs to be added to the colour or not. Doing this the atoms for the depth map will have flat colours.

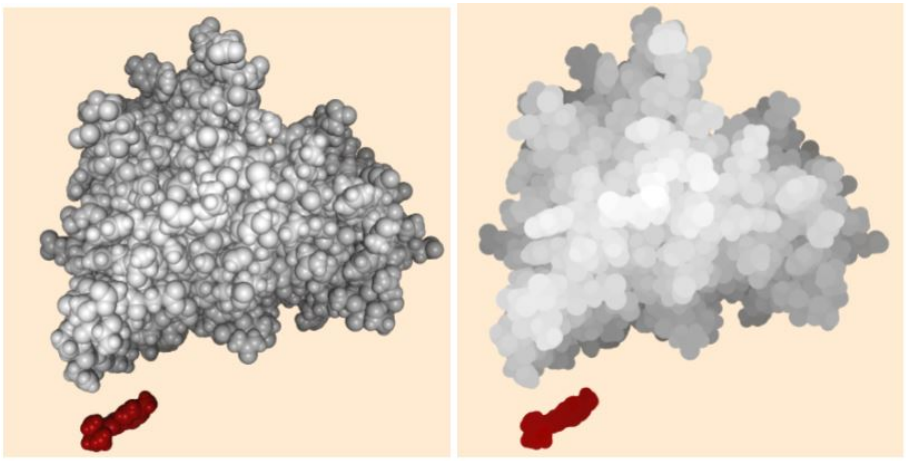


Figure 4.10: Comparison between not modified (left) shader and modified shader (right). The lighting makes more difficult to acquire the information fast

Next step is to be able to see where the ligand is no matter where it is. We could just draw the protein and afterwards the ligand over it and then compare the colors to know if the ligand is partially or totally visible from the viewpoint but we want the user to be able to identify these things as fast as possible. To do this, we paint the visible atoms in different tones of red and green if it is not visible from the viewpoint.

NGL Viewer has the makeImage() function that makes an image from what is shown in the viewer canvas in form of blob. First we capture an image of the ligand and then we load the molecule and capture a second one. These two images will be stored as blobs and later transformed into array buffers that will contain the PNG image. UPNG.js will then decode them into an array where every four values would correspond to one pixel. Finally we just need to compare both same position pixels; if the colour is the same for both pixels it means it is the background, the molecule or the ligand in front so we will put into a new image a pixel with the same values. Instead, if the values are different, it means one pixel contains the ligand and the other the molecule, so we will maintain the same gradient but we will change the colour to green. Once we have compared all pixels, we just encode the resultant image into blob that corresponds to a new PNG image and using FileSaver.js we download it with the name depthmap_x_y.png, where x corresponds to the epoch and y to the processor. Afterwards, we can use (if needed) a graphic converter to crop part of the edges and maximize the visible surface of the depthmap (always maintaining the aspect ratio).

Having selected one cluster, we just need to read which epoch and processor goes into each cell and load one of the images created aforehand.

4.4 Individual plot view

In chapter 3 and with the depths maps we have talked about small multiples but now we will a better explanation about how we designed ours.

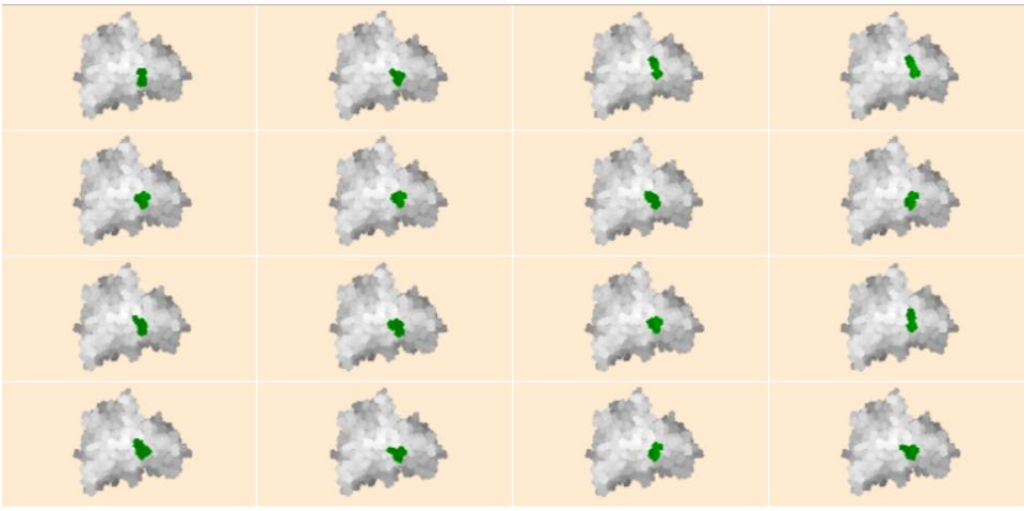


Figure 4.11: Example of Depth map view

4.4.1 Input

- run_trajectory_x.pdb
- run_report_x

4.4.2 Languages and libraries

- D3.js

4.4.3 Implementation

Small multiples

When talking about the cluster visualization view, there there was a line chart with the total energy of all paths that were in the four biggest clusters and four little ones where each will only show the paths in that specific cluster. This is good but if we want to compare one of the paths with others, that would be difficult.

Small multiples are a serie of small plots or graphs using the same scale and axes, where each contain a partition of the same dataset, making easier to compare them.

In this case, to reduce the data-ink ratio, we erase the labels of x and y-axis and values in the x-axis only because are the same as in the big plot in the cluster visualization view but we cannot delete the values of the axis in the y-axis. This is because the values are ranged from the minimum to the maximum value of total energy of that specific cluster.

Chapter 5

Interaction

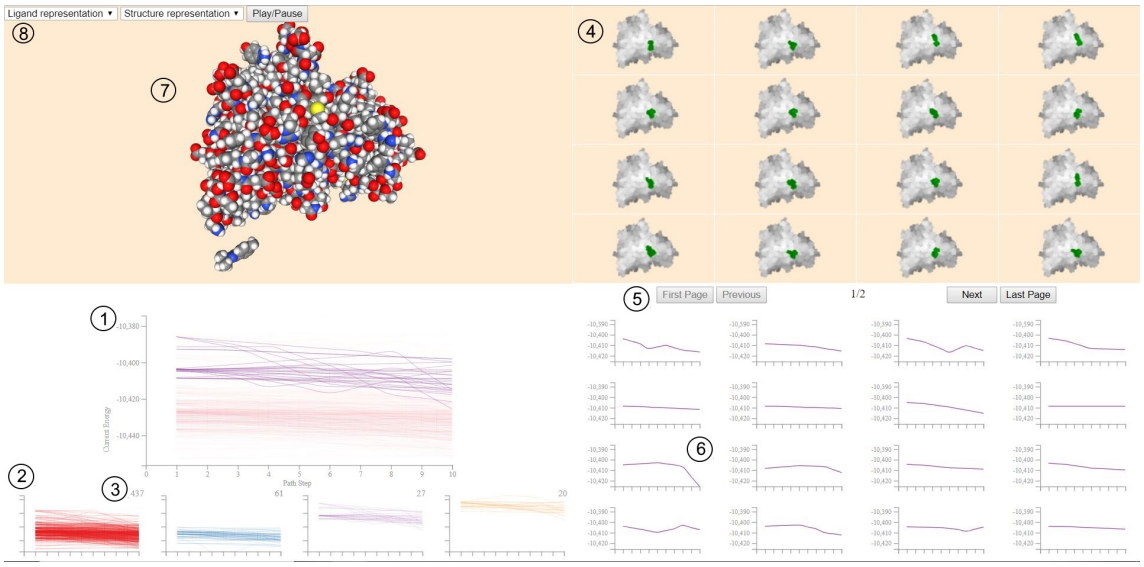


Figure 5.1: Comparison between not modified (left) shader and modified shader (right). The lighting makes more difficult to acquire the information fast

5.1 Cluster visualization view

1): The line chart contains all the paths of the four most populated clusters, being each one of them represented by a different colour. Using the mouse over the chart area, the user can interact with it; Using the left click, the user can drag and drop (press and move while holding down) to move around the plot. Using the scroll, the user can zoom in and out.

2): The four clusters data is distributed among four different line charts. When left clicking one of them, the depth map and individual chart views are updated with the chosen cluster. 1) will change too; the lines of the selected cluster will get thicker and the others will get semi-transparent. This help us to highlight the data while still maintaining the context.

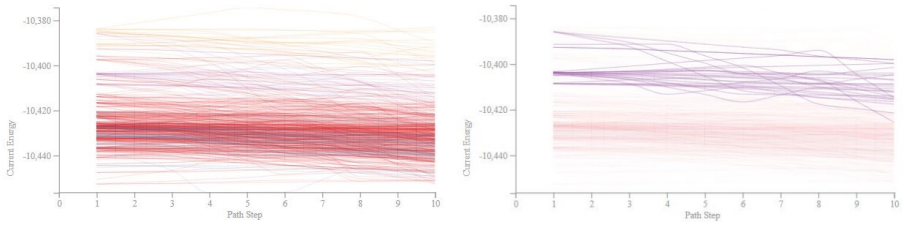


Figure 5.2: The plot on the left do not have any cluster selected unlike the one on the right. We can observe how the lavender lines are more visible.

3): Indicates the number of paths inside that cluster.

Once you presses any place inside the Cluster visualization view but outside 1) or any of 2), in case you have pressed 2), 1) will return to its initial state.

5.2 Depth map view

4): Once the user has selected a cluster, the paths are sorted in ascendent order depending on the mean total energy of the system (currrentEnergy). Each image represents one path and are positioned from left to right and from top to bottom. Pressing on any of the images will load that same path in the 3D viewer. The selected image and the equivalent individual chart will be highlighted in yellow. This let the user easily match the image with the chart and its correspondent 3D visualization.

If you stay on an image for a second, a tooltip will appear indicating from which epoch and processor is the image.

5): Page menu lets the user change pages so we can see the paths of the cluster that could be not seen initially. Changing page will affect the depth map view and the individual chart view. The number of the middle indicates the current page and the total number of pages. The buttons, as their texts say, First go to the first page, previous go one page back, next goes one page forward and last goes to the last page. The buttons will be disabled when there is no use for them (for example when there is only one page or for example the forward buttons when the user is in the last page.

5.3 Individual chart view

6): Each plot represents a path and all are sorted by mean current Energy of that path in ascendent order. Same interactions as 4); when pressing any of the plots both the plot and its corresponding depth map image will be highlighted in yellow and the 3D viewer will show its 3D model. Staying on the plot will make appear a tooltip showing epoch and processor.

5.4 3D viewer

7): Visualization 3D of the model corresponding to the depth map/line chart. The functionalities are given by NGL viewer. Some interactions are:

- Drag and drop with left click to rotate around the center of the escene

- Drag and drop with right click to translate the model.
- Scroll to zoom in and out
- 'I' Key to make it spin.
- 'R' Key to reset model position to initial.
- Left click on any atom will move the model so that atom is in the center.
- Putting the mouse over an atom will show the name. The name structure is "Atom: "+ [Residue name] + residue sequence number + chain identifier + atom name

8): The first two elements are dropdown buttons. The first changes the ligand representation and the second the protein representation. Both have the next representation:

- Ball-and-Stick: One of the most common representations where atoms are colored according to atom type and the bonds echo this information. The atoms are displayed as spheres and the bonds as cylinders. The aspect ratio is a number between 1 and 10 defining how much bigger the radius of the spheres is compared to the radius of the cylinders.
- Licorice: Variation of Ball-and-Stick where the aspect ratio is equal to 1 (both elements have the same radius).
- Spacefill: Atoms are displayed as a set of space-filling spheres.

Moreover, molecules have an extra representation:

- Ribbon: A thin ribbon is displayed along the main backbone trace.

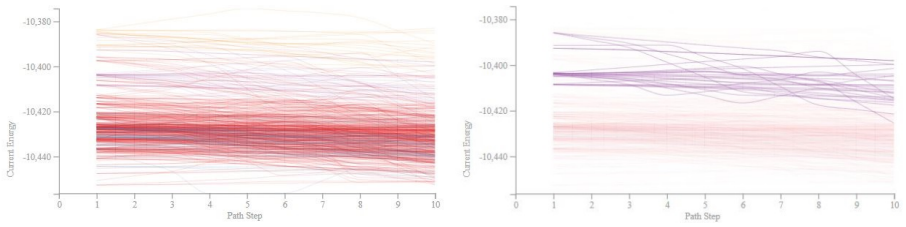


Figure 5.3: The plot on the left do not have any cluster selected unlike the one on the right. We can observe how the lavender lines are more visible.

The last button animates or pauses the animation of the path.

Chapter 6

Conclusions and future work

In this work we have presented a an exploratory visualization tool that is able to take a great number of paths and divide them into different patterns, at the same time that is able to remove outliers.

With the use of small simulations, the clustered paths are easier to differenciate and the fact that they are ordered by lowest energy helps to go to the ones that are more easy to do docking.

As future work we would like to implement using data from GROMACS, is a molecular dynamics package mainly designed for simulations of proteins, lipids, and nucleic acids, which has access to other type of data, as different types of energy.

Chapter 7

Bibliography

Cgl.ucsf.edu. (2019). Introduction to Protein Data Bank Format. [online] Available at: https://www.cgl.ucsf.edu/chime
[Accessed 27 Jun. 2019].

Website.education.wisc.edu. (2019). D3 - A Beginner's Guide to Using D3. [online] Available at: https://website.education.wisc.edu/ swu28/d3t/visualization.html [Accessed 27 Jun. 2019].

Website.education.wisc.edu. (2019). D3 - A Beginner's Guide to Using D3. [online] Available at: https://website.education.wisc.edu/ swu28/d3t/visualization.html [Accessed 27 Jun. 2019].

Website.education.wisc.edu. (2019). D3 - A Beginner's Guide to Using D3. [online] Available at: https://website.education.wisc.edu/ swu28/d3t/visualization.html [Accessed 27 Jun. 2019].